ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ Η/Υ.

ΕΠΙΤΑΧΥΝΣΗ ΚΩΔΙΚΑ ΠΕΠΕΡΑΣΜΕΝΩΝ ΣΤΟΙΧΕΙΩΝ ΓΙΑ ΜΗΧΑΝΟΛΟΓΙΚΕΣ ΕΦΑΡΜΟΓΕΣ

FINITE ELEMENTS CODE ACCELERATION FOR MECHANICAL ENGINEERING APPLICATIONS

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΣΠΥΡΙΔΩΝ Κ. ΘΕΡΜΟΣ

Επιβλέποντες Καθηγητές : Γεώργιος Σταμούλης
Καθηγητής ΤΗΜΜΥ

Νικόλαος Αράβας
Καθηγητής ΤΜΜ

Βόλος, Σεπτέμβριος 2013

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
& ΜΗΧΑΝΙΚΩΝ Η/Υ

ΕΠΙΤΑΧΥΝΣΗ ΚΩΔΙΚΑ ΠΕΠΕΡΑΣΜΕΝΩΝ ΣΤΟΙΧΕΙΩΝ ΓΙΑ ΜΗΧΑΝΟΛΟΓΙΚΕΣ ΕΦΑΡΜΟΓΕΣ

FINITE ELEMENTS CODE ACCELERATION FOR MECHANICAL ENGINEERING APPLICATIONS

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΣΠΥΡΙΔΩΝ Κ. ΘΕΡΜΟΣ

Επιβλέποντες Καθηγητές : Γεώργιος Σταμούλης
Καθηγητής ΤΗΜΜΥ

Νικόλαος Αράβας
Καθηγητής ΤΜΜ

Εγκρίθηκε από την διμελή εξεταστική επιτροπή την ….η Σεπτεμβρίου 2013

.....	.....
Γ. Σταμούλης	Ν. Αράβας
Καθηγητής ΤΗΜΜΥ	Καθηγητής ΤΜΜ

Διπλωματική Εργασία για την απόκτηση του Διπλώματος του Μηχανικού Η/Υ Τηλεπικοινωνιών και Δικτύων του Πανεπιστημίου Θεσσαλίας, στα πλαίσια του Προγράμματος Προπτυχιακών Σπουδών του Τμήματος Μηχανικών Η/Υ Τηλεπικοινωνιών και Δικτύων του Πανεπιστημίου Θεσσαλίας.

……..
Σπυρίδων Κ. Θερμός
Διπλωματούχος Μηχανικός Η/Υ Τηλεπικοινωνιών και Δικτύων Πανεπιστημίου Θεσσαλίας

Στην Ιορδάνα και την παρέα του Βόλου.

Ευχαριστίες

Με την περάτωση της παρούσας διπλωματικής εργασίας, θα ήθελα να ευχαριστήσω τους επιβλέποντες καθηγητές κ. Σταμούλη Γεώργιο και κ. Νικόλαο Αράβα για την εμπιστοσύνη που επέδειξαν στο πρόσωπό μου και την συνεχή καθοδήγηση που διευκόλυνε την εκπόνηση της συγκεκριμένης εργασίας.
Ακόμα, θα ήθελα να ευχαριστήσω τους Κωνσταντή Νταλούκα και Μπάμπη Αντωνιάδη για την υποστήριξη και τις ουσιώδεις παρεμβάσεις τους.
Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου για την αμέριστη υποστήριξη και πολύτιμη βοήθεια που μου παρείχαν τόσο κατά την διάρκεια των σπουδών μου όσο και κατά την εκπόνηση της διπλωματικής αυτής εργασίας.

Θερμός Σπύρος

Βόλος 2013

Περίληψη

Τα δομικά στοιχεία των μηχανολογικών κατασκευών υπόκεινται σε επαναλαμβανόμενες κυκλικές καταπονήσεις, από τις οποίες δημιουργούνται και διαδίδονται ρωγμές. Ο υπολογισμός διάδοσης μιας ρωγμής ανάλογα με τον τύπο της, είναι μία διαδικασία που περιλαμβάνει μια σειρά από πράξεις πινάκων οι οποίες υλοποιούνται με τη βοήθεια της τεχνολογίας. Οι περισσότερες εφαρμογές που υπολογίζουν τη διάδοση της ρωγμής είναι γραμμένες σε γλώσσα FORTRAN, μια γλώσσα σχεδιασμένη αποκλειστικά για μηχανικούς που θέλουν να μοντελοποιήσουν και να επιλύσουν προβλήματα που αφορούν σε μηχανολογικές κατασκευές. Όσο μεγαλύτερα και πιο πολύπλοκα είναι τα προβλήματα αυτά τόσο πιο χρονοβόροι είναι οι υπολογισμοί.

Τα τελευταία χρόνια παράλληλα, έγιναν άλματα στην τεχνολογία έτσι ώστε να μπορούν να υποστηριχτούν υπολογισμοί που επιλύουν απαιτητικά, σε πράξεις, προβλήματα και να ελαχιστοποιηθεί ο χρόνος επίλυσής τους. Οι GPUs είναι πλατφόρμες με μεγάλο αριθμό πυρήνων οι οποίοι πραγματοποιούν υπολογισμούς παράλληλα με πολύ καλή απόδοση και είναι ότι καλύτερο έχει να επιδείξει αυτή τη στιγμή η βιομηχανία για να βοηθήσει στην γρηγορότερη εκτέλεση των εφαρμογών που αναφέρθηκαν παραπάνω.

Στόχος αυτής της διπλωματικής εργασίας είναι να παρουσιάσει μια εφαρμογή που υπολογίζει τη διάδοση ρωγμής mode III, να κάνει μια εισαγωγή στην νέα τεχνολογία που επικρατεί τελευταία στον τομέα των υπολογισμών και να αποδείξει ότι παραλληλοποιώντας τη συγκεκριμένη εφαρμογή, είναι δυνατή η αισθητή επιτάχυνσή της.

Abstract

The modules of mechanical constructions underlie recurrent cyclic distresses, from which fissures are brought about and disseminated. The computation of a fissure's dissemination commensurate to its type is a process that includes a series of matrix operations that are carried into effect via technology. Most applications that compute the dissemination of the fissure are written in FORTRAN, a language designed exclusively for mechanical engineers who wish to model and solve problems relative to mechanical constructions. The bigger and the complicated the problems are, the more time-consuming the computations are.

Simultaneously, during the past few years, great strides have been made in technology so that the computations that solve exacting problems can be asserted, in acts, thus the time consumed to solve them can be reduced. The GPUs are chips with great number of cores that materialize computations in parallel while carrying out a very good execution and therefore being the best there is to show, in the industry, that help with the fastest execution of the above mentioned applications.

The aim of this diploma thesis is to present an application that computes the mode III fissure dissemination, to make an introduction in the new technology dominant lately in the field of computations and to prove that by parallelizing this application, its serious acceleration is possible.

Περιεχόμενα

Table of Contents

Συντομογραφίες

CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
FORTRAN	Formulae Translator
GPU	Graphics Processing Unit
ILP	Instruction Level Parallelism
LAPACK	Linear Algebra Package
OPENCL	Open Computing Language
SM	Streaming Multiprocessor

1. Εισαγωγή

1.1 Παρουσίαση του προβλήματος και συμβολή της εργασίας

Πολλές φορές για να επιλύσουμε ένα πρόβλημα που απαιτεί υπολογισμούς όχι τόσο πολύπλοκους αλλά με πολύ μεγάλο αριθμό στοιχείων δεν αρκεί στυλό και χαρτί. Ένας μηχανικός αντιμετωπίζει συχνά τέτοια προβλήματα και καταφεύγει στη δημιουργία εφαρμογών για την επίλυση αυτών των προβλημάτων, οι οποίες χρησημοποιούν τις υπολογιστικές μονάδες ενός επεξεργαστή. Οι περισσότερες μηχανολογικές εφαρμογές είναι σε γλώσσα Fortran, μια γλώσσα εξιδανικευμένη για μαθηματικούς υπολογισμούς. Είναι όμως και η πιο γρήγορη υλοποίηση; Όσο μεγαλύτερος ο όγκος των στοιχείων που έχουμε για ένα πρόβλημα τόσο περισσότερος και ο χρόνος που πρέπει να σπαταλήσουμε για την επίλυσή του.

Η δημιουργία και διάδωση μιας ρωγμής σε κάποιο υλικό είναι ένα υπαρκτό πρόβλημα και ο υπολογισμός διάδοσης ρωγμής είναι μια χρονοβόρα υπολογιστική διαδικασία μιας και έχει πολλά στοιχεία που εμπλέκονται στην επίλυση του προβλήματος.

Η εργασία έχει ως σκοπό την μελέτη της απόδοσης στον χρόνο εκτέλεσης μιας εφαρμογής που υπολογίζει το διάδοση ρωγμής, αν μεταφερθούν τα πιο χρονοβόρα τμήματα αυτής στην κάρτα γραφικών (GPU) και τη σύγκριση των αποτελεσμάτων της με τα αντίστοιχα της κεντρικής μονάδας επεξεργασίας (CPU). Έχει παρατηρηθεί τα τελευταία χρόνια ότι η χρήση των καρτών γραφικών (GPU) σαν επιταχυντών, έχει συμβάλλει σημαντικά στην μείωση του χρόνου εκτέλεσης πολλών μεγάλων και υπολογιστικά χρονοβόρων εφαρμογών. Οι δυνατότητες που προσφέρουν οι προαναφερόμενες κάρτες όσον αφορά στις μονάδες εκτέλεσης που μπορούν να τρέξουν παράλληλα πάνω σε αυτές είναι πολύ μεγαλύτερες σε σχέση με αυτές που παρέχει η κεντρική μονάδα επεξεργασίας (CPU) και αυξάνονται με την πάροδο του χρόνου. Έχουν αναπτυχθεί διάφορα μοντέλα παραλληλισμού για την αξιοποίησή τους, με πιο διαδεδομένα

αυτό της CUDA και του OpenCL. Το μοντέλο CUDA ανήκει στην NVIDIA, ενώ αυτό του OpenCL ανήκει σε μια πληθώρα εταιριών κατασκευής υλικού, στις οποίες εντάσσεται και η NVIDIA. Στην παρούσα μελέτη μας, θα χρησιμοποιηθεί το μοντέλο της CUDA.

1.2 Διάρθρωση της διπλωματικής εργασίας

Αρχικά, στη διπλωματική αυτή εργασία, θα δούμε ποιο ακριβώς είναι το πρόβλημα με το οποίο καταπιάνεται η εφαρμογή μας, αλλά και ποια γλώσσα χρησημοποιήθηκε για την δημιουργία της εφαρμογής αυτής.

Στη συνέχεια, παρουσιάζεται η μετατροπή της εφαρμογής σε μια άλλη γλώσσα προγραμματισμού. Η μετατροπή αυτή μας επιτρέπει να εξετάσουμε τον χρόνο που απαιτείται για να ολοκληρωθούν οι υπολογισμοί όταν η εφαρμογή αυτή τρέξει σε έναν πυρήνα επεξεργαστή ή και σε περισσότερους πυρήνες τις ίδιας CPU.

Αμέσως μετά γίνεται παρουσίαση των μονάδων GPU αλλά και μια εισαγωγή στη γλώσσα CUDA, η οποία θα μας επιτρέψει να παραλληλοποιήσουμε την εφαρμογή μας, όπου αυτό είναι δυνατό, και να γλιτώσσουμε αρκετό χρόνο σε σχέση με τη σειριακή υλοποίηση.

Τέλος, στην ενότητα των πειραματικών αποτελεσμάτων θα δούμε στην πράξη τους χρόνους που καταναλώνουν οι υπολογισμοί αλλά και το πρόγραμμα συνολικά και θα έχουμε την ευκαιρία να βγάλουμε συμπεράσματα συγκρίνοντας τους χρόνους ολοκλήρωσης μεταξύ διαφορετικών υλοποιήσεων.

2. Εφαρμογή

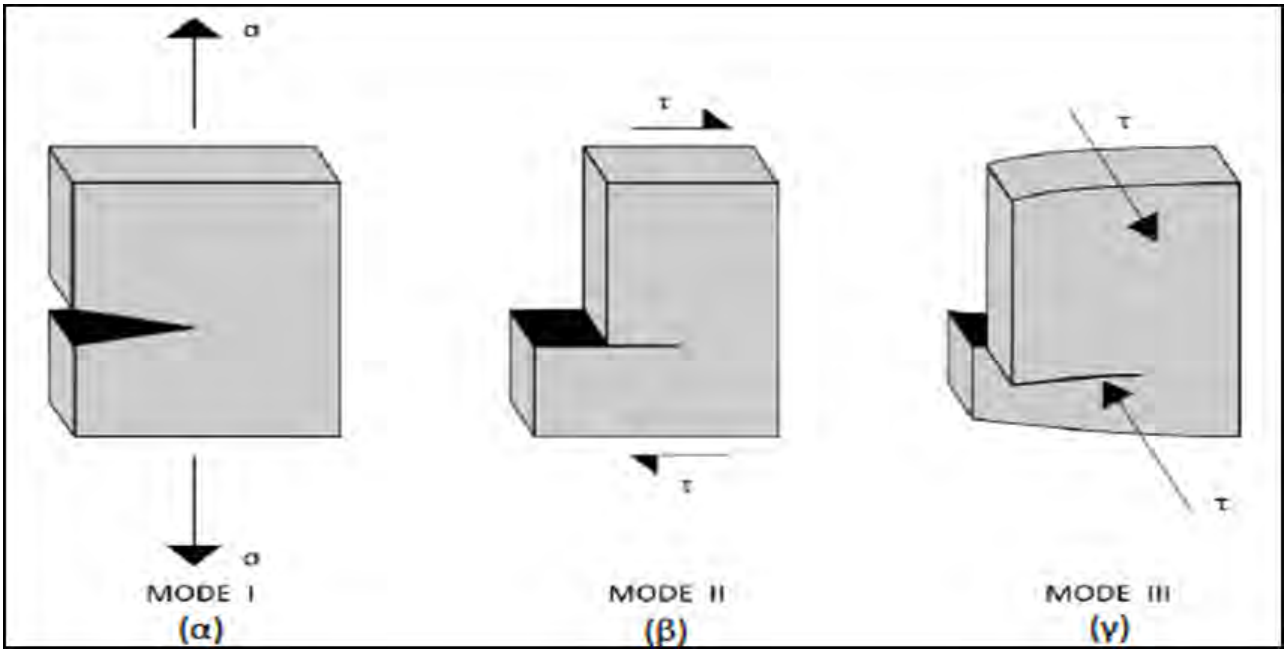
2.1 Δημιουργία και Διάδοση Ρωγμών

Ρωγμή στο επίπεδο μπορεί να ορισθεί ότι είναι το όριο ελλειπτικής οπής, ο μικρός ημιάξονας της οποίας τείνει στο μηδέν ή ότι είναι μια φυσική ασυνέχεια χωρίς πάχος (μαθηματική γραμμή).
Ο G.R. Irwin μελετώντας το τασικό πεδίο στην αιχμή μιας ρωγμής παρατήρησε ότι υπάρχουν τρεις ανεξάρτητοι τρόποι μετακινήσεως (παραμορφώσεως) των χειλέων της ανάλογα με το είδος της εφαρμοζόμενης φορτίσεως. Οποιαδήποτε άλλη μορφή παραμορφώσεως μπορεί να θεωρηθεί ότι προκύπτει από το συνδυασμό των τριών αυτών τύπων.

α) Επίπεδος Εφελκυστικός Τύπος-I (Opening Mode ή Mode I): Αντιστοιχεί σε ομοαξονικό εφελκυσμό κάθετο στο μεγάλο άξονα της ρωγμής. Στην περίπτωση αυτή τα χείλη της ρωγμής τείνουν να διαχωριστούν συμμετρικά ως προς το πριν την παραμόρφωση επίπεδό της. (Σχ. 1 α)

β) Συνεπίπεδος Διατμητικός Τύπος-II (Sliding Mode ή Mode II): Αντιστοιχεί σε διάτμηση κατά μήκος του μεγάλου άξονα της ρωγμής και προκύπτει όταν τα απέναντι χείλη της τείνουν να ολισθήσουν το ένα σχετικά με το άλλο προς αντίθετες διευθύνσεις, αλλά πάνω στο ίδιο επίπεδο. (Σχ. 1 β)

γ) Εγκάρσιος Διατμητικός Τύπος-III (Tearing Mode ή Mode III): Αντιστοιχεί στην εγκάρσια διάτμηση και προκύπτει όταν τα χείλη της ρωγμής διαχωρίζονται προς αντίθετες εγκάρσιες διευθύνσεις εξαιτίας της επιδράσεως ίσων και αντίθετων δυνάμεων κάθετων στο επίπεδο του σώματος. (Σχ. 1 γ).



Σχήμα 1 Είδη Ρωγμών

Η συγκεκριμένη εφαρμογή που βελτιστοποιείται σε αυτή την εργασία υπολογίζει τη διάδοση ρωγμής και συγκεκριμένα της ρωγμής Mode III.

2.2 FORTRAN

Η γλώσσα Fortran (από τα αρχικά FORmulae TRANslator - μεταφραστής τύπων) είναι μία από τις πρώτες γλώσσες προγραμματισμού υψηλού επιπέδου, η οποία χρησιμοποιήθηκε κυρίως σε επιστημονικές αλλά και σε εμπορικές εφαρμογές. Δημιουργήθηκε τη δεκαετία του 1950 από την IBM και χρησιμοποιείται μέχρι και σήμερα. Αρχικά η Fortran ήταν προσανατολισμένη στην επίλυση μαθηματικών προβλημάτων. Η εφαρμογή που εξετάζεται σε αυτή την εργασία είναι γραμμένη σε Fortran77. Γενικά η συγκεκριμένη γλώσσα προγραμματισμού έχει καλύτερη απόδοση σε σχέση με άλλες γλώσσες (για παράδειγμα C ή C++) όταν αντιμετωπίζουμε

ένα πρόβλημα με πράξεις μεταξύ πινάκων προκαθορισμένων διαστάσεων και υποστηρίζει πράξεις μονής και διπλής ακρίβειας.

2.3 Βιβλιοθήκη LAPACK

Η LAPACK είναι μια βιβλιοθήκη λογισμικού για αριθμητική γραμμική άλγεβρα. Παρέχει ρουτίνες για την επίλυση συστημάτων γραμμικών εξισώσεων και γραμμικών ελαχίστων τετραγώνων, προβλήματα ιδιοτιμών και ιδιαζουσών τιμών. Περιλαμβάνει επίσης ρουτίνες για την εφαρμογή των σχετικών παραγοντοποιήσεων πινάκων όπως LU, QR, Cholesky και Schur decomposition. Η LAPACK γράφτηκε αρχικά στη FORTRAN 77, αλλά μεταφέρθηκε στη Fortran 90 στην έκδοση 3.2 (2008). Οι ρουτίνες της μπορούν να χειριστούν απλούς και σύνθετους πίνακες με πράξεις απλής και διπλής ακρίβειας.

Στην εφαρμογή μας χρησημοποιείται μία συγκεκριμένη ρουτίνα της LAPACK , η DGBSV. Η ρουτίνα αυτή υπολογίζει τη λύση για ένα πραγματικό σύστημα γραμμικών εξισώσεων A*X = B όπου A είναι ένας band πίνακας τάξης N, με KL υποδιαγώνια και KU υπερδιαγώνια στοιχεία, ενώ X και B είναι N x NRHS πίνακες. Η LU παραγοντοποίηση, με μερική αντικατάσταση και εναλλαγές γραμμών, χρησημοποιείται για να εκφραστεί ο πίνακας A ως A = L*U, όπου L είναι ένας κάτω τριγωνικός πίνακας με KL υποδιαγώνια στοιχεία και U ένας άνω τριγωνικός πίνακας με KL+KU υπερδιαγώνια στοιχεία. Η παραπάνω μορφή του A χρησημοποιείται για την επίλυση του συστήματος των εξισώσεων της A*X = B.

Πιο αναλυτικά, η ρουτίνα:

SUBROUTINE DGBSV (N, KL, KU, NRHS, AB, LDAB, IPIV, B, LDB, INFO)

Όπου INFO, KL, KU, LDAB, N, NRHS ακέραιοι, IPIV πίνακας ακεραίων και AB, B πίνακες διπλής ακρίβειας. Ας δούμε όμως και λίγο περισσότερο τα ορίσματα,

1. *N (είσοδος) ακέραιος, ο αριθμός των γραμμικών εξισώσεων, N>=0.*

2. *KL (είσοδος) ακέραιος, ο αριθμός των υποδιαγώνιων στοιχείων του band A, KL>=0.*

3. *KU (είσοδος) ακέραιος, ο αριθμός των υπερδιαγώνιων στοιχείων του band A, KU>=0.*

4. *NRHS (είσοδος) ακέραιος, ο αριθμός των στηλών του πίνακα B, NRHS>=0.*

5. *AB (είσοδος/έξοδος) πίνακας διπλής ακρίβειας, διαστάσεων (LDAB, N). Κατά την είσοδο, ο πίνακας A σε band storage, στις σειρές KL +1 μέχρι 2* KL+KU+1. Η j-στη στήλη του A αποθηκεύεται στη j-στη στήλη του πίνακα AB ως εξής: AB (KL + KU+1+ i - j, j) = A (i, j) για max(1, j-KU) <= i <= min (N, j + KL).*
   *Κατά την έξοδο, λεπτομέρειες της παραγοντοποίησης: ο U αποθηκεύεται ως ένας άνω τριγωνικός band πίνακας με KL + KU + 1 υπερδιαγώνια στοιχεία στις σειρές 1 έως KL + KU +1, και οι πολλαπλασιαστές που χρησιμοποιούνται κατά τη διάρκεια της παραγοντοποίησης αποθηκεύονται στις σειρές KL + KU + 2 έως 2*KL+KU+1.*

6. *LDAB (είσοδος) ακέραιος, ο αριθμός γραμμών του πίνακα AB, LDAB>=2*KL+KU+1.*

7. *IPIV (είσοδος) πίνακας ακεραίων, διάσταση (N), χρησημοποιείται για εναλλαγές και μεταθέσεις.*

8. *B (είσοδος/έξοδος) πίνακας διπλής ακρίβειας, διαστάσεων (LDB, NRHS). Κατά την είσοδο είναι ο NxNRHS πίνακας B στο δεξιό μέλος του συστήματος. Κατά την έξοδο (αν INFO = 0)είναι ο NxNRHS πίνακας X που είναι και η λύση του συστήματος.*

9. *LDB (είσοδος) ακέραιος, ο αριθμός γραμμών του πίνακα B, LDB >= max (1, N).*

10. *INFO (έξοδος) ακέραιος, αν είναι ίσο με 0 τότε επιτυχείς έξοδος. Αν είναι μικρότερο του 0 (INFO = -i) τότε το i-στο όρισμα έχει μη αποδεκτή τιμή. Αν είναι μεγαλύτερο του 0 (INFO = i) τότε το U(i, i)=0, η παραγοντοποίηση έχει γίνει αλλά η λύση δεν έχει υπολογιστεί.*

3 Μετατροπή σε γλώσσα C

Η μετατροπή του κώδικα της Fortran σε γλώσσα C έγινε για να έχουμε μια πιο ευανάγνωστη εφαρμογή η οποία θα μεταγλωττιστεί με σύγχρονους μεταγλωττιστές με σημαντικές προοπτικές βελτιστοποίησης αφήνοντας πίσω τους παλιότερους μεταγλωττιστές της Fortran αλλά και τους αυστηρούς κανόνες σύνταξης της γλώσσας που δεν αφήνουν και πολλά περιθώρια βελτιστοποίησης, τουλάχιστον σε σύγκριση με την παραλληλοποίηση που θα παρουσιάσουμε αργότερα.

3.1 Ενσωμάτωση της LAPACK

Η αρχική σκέψη ήταν ο κώδικας της C να γραφτεί στο εργαλείο της Microsoft το Microsoft Visual Studio και να προστεθούν εκεί όλα τα αρχεία που θα μας επέτρεπαν να χρησημοποιήσουμε την βιβλιοθήκη LAPACK. Στην πορεία, και μετά από αρκετή σκέψη, ο κώδικας C γράφτηκε σε UNIX όπου τα αντίστοιχα πακέτα που επιτρέπουν την χρήση της LAPACK σε ένα πρόγραμμα C είναι πιο εύκολο να εγκατασταθούν και να συνδεθούν. Τα πακέτα που χρησημοποιήθηκαν είναι τα :

- liblapack-dev
- liblapack3gf

Με την εγκατάσταση των πακέτων θεωρητικά αρκεί η προσθήκη της εντολής –llapack κατά την μεταγλώττιση έτσι ώστε να μπορούμε να κάνουμε χρήση της DGBSV( ) που χρειαζόμαστε στην εφαρμογή μας.

Όμως ο μεταγλωττιστής gcc δεν αναγνωρίζει το κομμάτι του προγράμματος όπου καλείται η DGBSV( ). Με την προσθήκη τριών εντολών που εμφανίζονται παρακάτω και την χρήση του g++ αυτή τη φορά ο μεταγλωττιστής καταλαβαίνει την κλήση μιας ρουτίνας της βιβλιοθήκης LAPACK και έτσι πρακτικά η υλοποίηση μπορεί να συνεχιστεί.

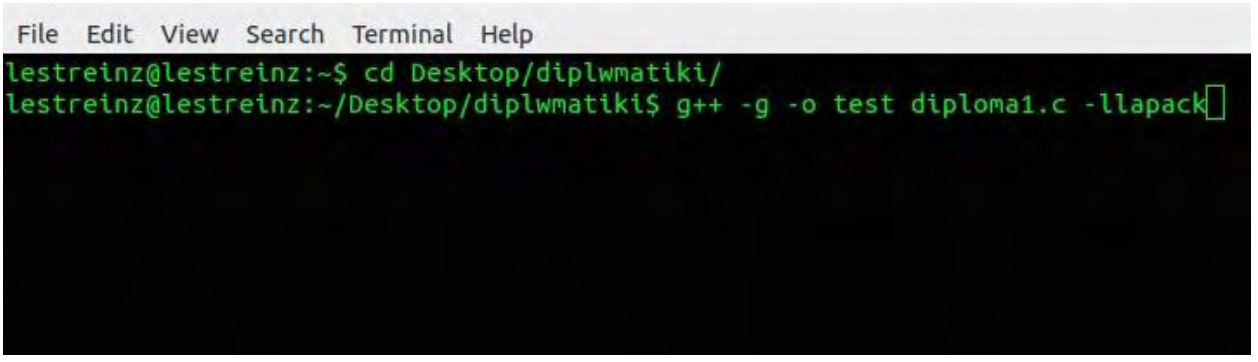
```
#define _cplusplus

#ifdef _cplusplus
extern "C" void dgbsv_(int &N, int &KL, int &KU, int &NRHS, double *AB, int &LDAB, int *IPIV, double *B, int &LDB, int &INFO);
#endif
```

Σχήμα 2 Απόσπασμα Κώδικα μετά την Μετατροπή

3.2 Μεταγλώττιση

Για τη μεταγλώττιση του προγράμματος, συμπεριλαμβανομένης της συνάρτησης DGBSV( ), όπως τμηματικά αναφέρθηκε παραπάνω χρησιμοποιήθηκε η εντολή :

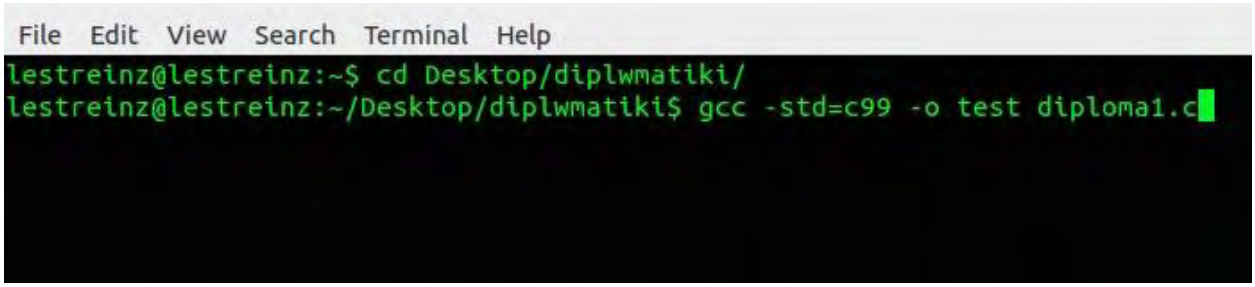


Σχήμα 3 Εντολή από τερματικό για C++

Όμως για να είναι επιτυχής η μεταγλώττιση του προγράμματος έπρεπε να αλλάξουν κάποιες συναρτήσεις που αφορούν κυρίως σε τρι-

γωνομετρία, άλγεβρα και να προστεθούν αντίστοιχες βιβλιοθήκες. Για παράδειγμα η εντολή ** που στη Fortran αντιστοιχεί στην ύψωση σε δύναμη, στη C πρέπει να χρησημοποιήσουμε την συνάρτηση pow( ) της βιβλιοθήκης math.

Αν παρατηρήσουμε την εφαρμογή βλέπουμε ότι στην πραγματικότητα το πρόγραμμα είναι μια αρκετά μεγάλη προεργασία που περιλαμβάνει πράξεις μεταξύ πινάκων και όχι μόνο, με σκοπό την δημιουργία των εισόδων για την ρουτίνα DGBSV( ) που επιλύει το σύστημά μας στο τέλος. Δηλαδή, μπορούμε να δούμε το πρόγραμμά μας σαν δύο υπο-προγράμματα που το πρώτο είναι υλοποιημένο σε C και το δεύτερο είναι μια κλήση ρουτίνας. Οπότε αργότερα, όταν μιλήσουμε για βελτι-στοποίηση, θα αναφερθούμε στο πρώτο υποπρόγραμμα το οποίο μπορεί να μελετηθεί ξεχωριστά και να παραλληλοποιηθεί (μέρος αυτού). Με τη λογική ότι θα χρειαστούμε να μελετήσουμε χρόνους σε αυτό το κομμάτι του προγράμματος και για την μελέτη αυτή θα χρησημοποιήσουμε τον GNU profiler του gcc, μπορούμε να μεταγλωττίσουμε ξεχωριστά το υποπρόγραμμα αυτό. Η εντολή για την μεταγλώττιση αυτή είναι :



Σχήμα 4 Εντολή απ'ο τερματικό για C

3.3 Δέσμευση Μνήμης

Η Fortran γενικότερα δουλεύει να προκαθορισμένα μεγέθη πινάκων. Όμως, στη C μας δίνεται η δυνατότητα της δυναμικής δέσμευσης μνήμης. Χρησημοποιώντας την εντολή malloc μπορούμε να δεσμεύσουμε δυναμικά χώρο στη μνήμη και να περνάμε το εκάστοτε μέγεθος των πινάκων που θέλουμε σαν ορίσματα κατα το τρέξιμο αλλά και σαν πληροφορία από το αρχείο εισόδου που θα έχουμε κάθε φορά. Έτσι ουσιαστικά κάνουμε την εφαρμογή μας πιο ευέλικτη και εύχρηστη.

4 Παραλληλοποίηση

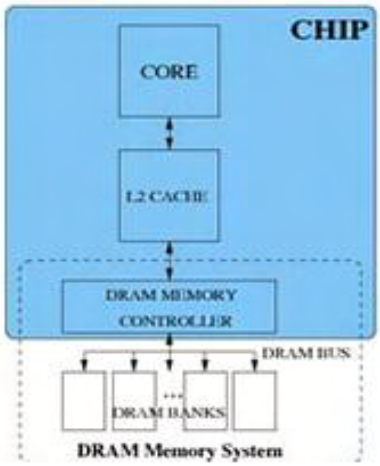
Η συνολική προεργασία που γίνεται για να δημιουργηθούνε τα ορίσματα της συνάρτησης DGBSV( ) έτσι ώστε να γίνει η επίλυση του συστήματος A*X = B είναι ουσιαστικά πράξεις μεταξύ πινάκων που εκτελούνται σειριακά. Στόχος αυτής της διπλωματικής εργασίας είναι να επιταχυνθεί αυτή η προεργασία, κάτι που θα μας οδηγήσει στην επιτάχυνση της εφαρμογής στο σύνολό της.

4.1 Γενικά

4.1.1 Εποχή SINGLE – CORE

Την περασμένη δεκαετία ο πρωταρχικός στόχος της βιομηχανίας υπολογιστών ήταν η απόδοση. Την μέγιστη αυτή απόδοση προσπαθούσαν

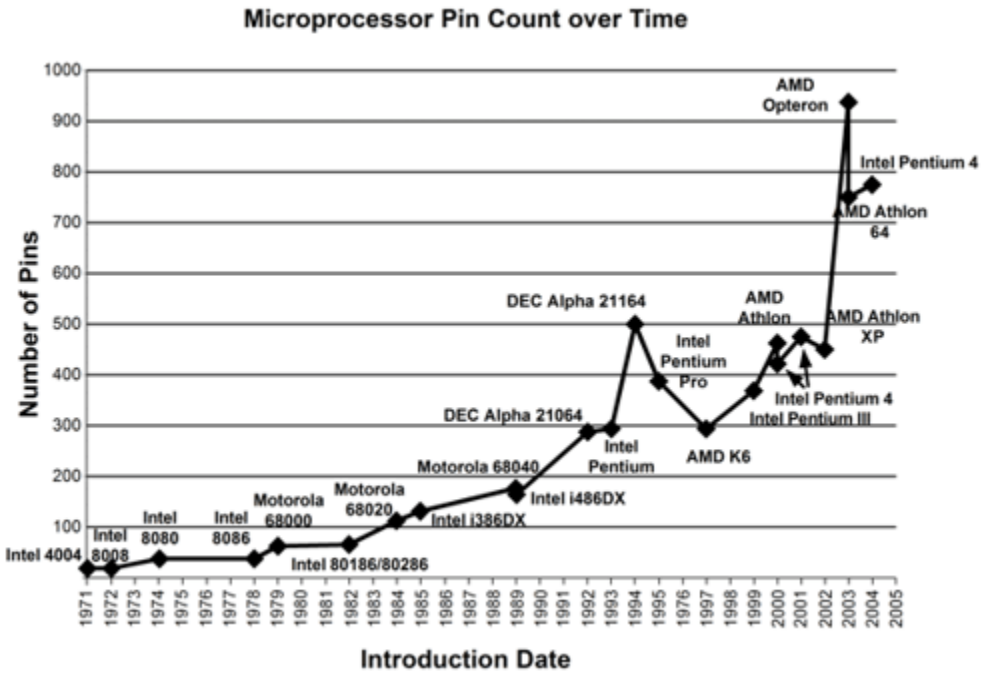
να την πετύχουν βάζοντας όλο και περισσότερα transistors πάνω στο chip, αυξάνοντας τη συχνότητα του ρολογιού και δημιουργώντας καινούργιες βελτιστοποιημένες αρχιτεκτονικές.



Σχήμα 5 Αρχιτεκτονική Απλού Πυρήνα

Επιβεβαιώνοντας το νόμο του Moore, ο οποίος προέβλεψε ότι ο αριθμός των transistor πάνω στο chip θα διπλασιάζονταν κάθε 18 μήνες, η πυκνότητα των transistors αυξήθηκε κατά πολύ τα τελευταία σαράντα χρόνια και η απόδοση μεγάλωνε συνεχώς. Όμως, βάζοντας όλο και περισσότερα transistors, εκτός από την απόδοση αυξήθηκε και η κατανάλωση ρεύματος αλλά και η θερμότητα η οποία από ένα σημείο και μετά θα μπορούσε να καταστρέψει το chip. Έτσι, εκτιμήθηκε ότι κάποια στιγμή θα έπρεπε να κρατάμε κάποια transistors σβηστά (dark) όταν κάποια άλλα θα χρησημοποιούνται για διάφορες λειτουργίες (φαινόμενο dark silicon). Επίσης, η θεωρία ότι όσο μεγαλύτερη συχνότητα έχει ο επεξεργαστής τόσο πιο αποδοτικός είναι έφτασε το άνω όριο μιας και από ένα σημείο και πάνω όσο και να αυξηθεί η συχνότητα, η απόδοση παραμένει ίδια. Τέλος, η περαιτέρω βελτιστοποίηση του ILP έγινε σχεδόν αδύνατη. Οπότε αντί να έχουμε μικρότερους, αποδοτικότερους

και λιγότερο απαιτητικούς σε κατανάλωση ρεύματος πυρήνες φτάσαμε να έχουμε θερμότερους και πολύ απαιτητικούς.

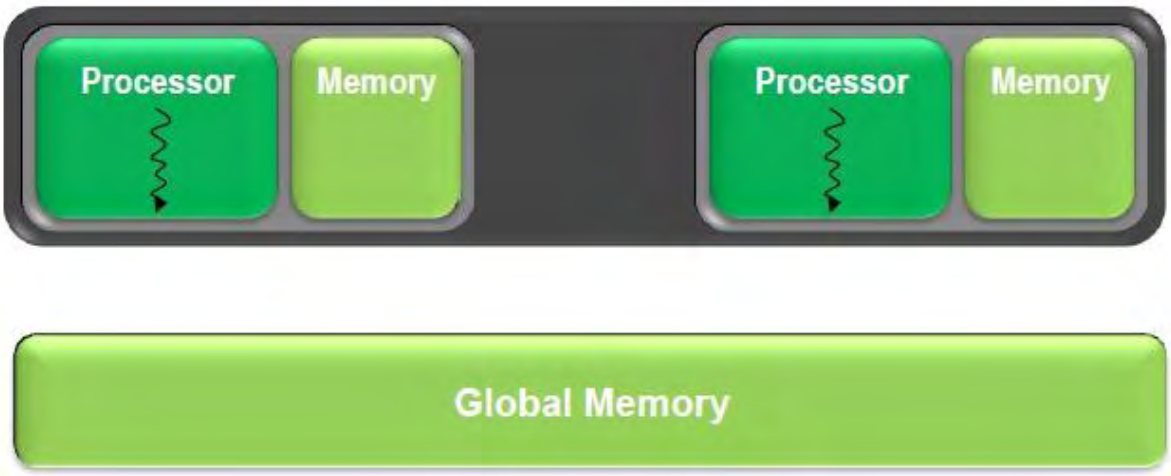


Σχήμα 6 Σχέση transistor-on-chip και έτους

4.1.2 Εποχή MULTICORE

Επόμενο ήταν να κυριαρχήσει μια νέα ιδέα που έλυνε τα παραπάνω προβλήματα και συνέχιζε να αυξάνει την απόδοση. Η ιδέα αυτή ήταν να χρησημοποιηθούν περισσότεροι πυρήνες σε έναν επεξεργαστή οι οποίοι θα χρονίζονται σε μικρότερες συχνότητες αλλά θα εκτελούν εντολές παράλληλα, ανεβάζοντας την απόδοση. Οι χαμηλές συχνότητες αυτές μείωσαν κατά πολύ την κατανάλωση ενέργειας αλλά προσωρινά. Ο αριθμός των transistors συνεχίζει να αυξάνεται και το φαινόμενο dark silicon που

αναφέρθηκε και παραπάνω είναι ορατό. Ακόμα, οι μετάβαση από τον ένα πηρύνα στους περισσότερους άλλαξε τη φιλοσοφία των προγραμματιστών που τώρα δεν προσπαθούν να βελτιστοποιήσουν τον σειριακό κώδικα, αλλά προσπαθούν να παραλληλοποιήσουν κομμάτια της εφαρμογής έτσι ώστε να έχουν καλύτερο συνολικό χρόνο υλοποίησης.



Σχήμα 7 Κάθε πυρήνας ένα thread.

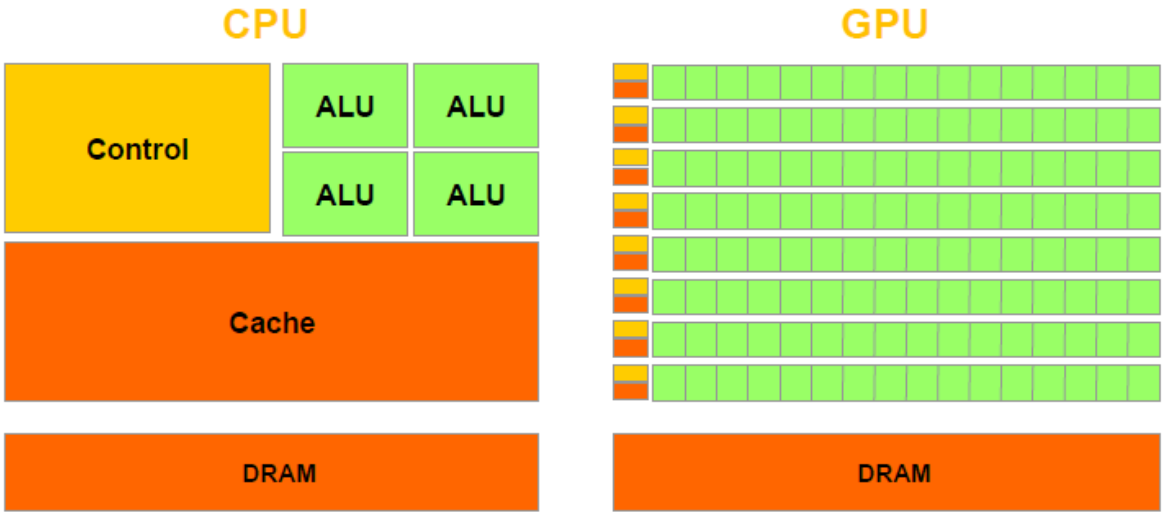
4.2 MANYCORES

4.2.1 GPUs - CUDA

Η GPU είναι ένα manycore σύστημα το οποίο αποτελείται από μεγάλο αριθμό πυρήνων.

Οι κάρτες γραφικών αρχικά σχεδιάστηκαν σαν συσκευές για την επεξεργασία και την οπτικοποίηση γραφικών του υπολογιστή. Όταν το 1990 το υλικό (hardware) έγινε προγραμματίσιμο, η εταιρία κατασκευής υλικού NVIDIA εκμεταλλεύτηκε αυτή την ευκαιρία και εφάρμοσε τη νέα

αυτή ικανότητα του hardware πάνω στις κάρτες γραφικών. Στη συνέχεια, το 1999, επινόησε τον όρο GPU (Graphics Processing Unit) και μια νέα εποχή, αποκαλούμενη ως GPGPU (General Purpose GPU), μόλις είχε ξεκινήσει. Η πρώτη επίσημη λύση της εν λόγω εταιρίας για υποστήριξη κώδικα εκτέλεσης σε GPUs δημοσιεύθηκε το 2006, όπου έγινε και η αποκάλυψη του μοντέλου παράλληλου προγραμματισμού CUDA (Compute Unified Device Architecture). Με την χρήση των δυνατοτήτων παράλληλης εκτέλεσης που προσφέρει η αρχιτεκτονική των GPUs, το εν λόγω μοντέλο μπορεί να αυξήσει πολύ έντονα την απόδοση των υπολογισμών που πραγματοποιούνται στην κάρτα γραφικών, σημειώνοντας σημαντικά μικρότερους χρόνους εκτέλεσης αυτών. Σε αυτό συμβάλλει η επικοινωνία που γίνεται ανάμεσα σε GPU και CPU (Central Processing Unit), έτσι ώστε να ανατεθεί η επίπονη χρονικά επεξεργασία των δεδομένων από την CPU στην κάρτα γραφικών. Μέσω αυτού, παρατηρείται και αποδεικνύεται ότι είναι προτιμότερη η ύπαρξη πολλών παράλληλων μονάδων εκτέλεσης χαμηλής ταχύτητας επεξεργασίας (GPU threads) από εκείνη των ελάχιστων παράλληλων μονάδων εκτέλεσης υψηλής ταχύτητας επεξεργασίας (CPU threads).



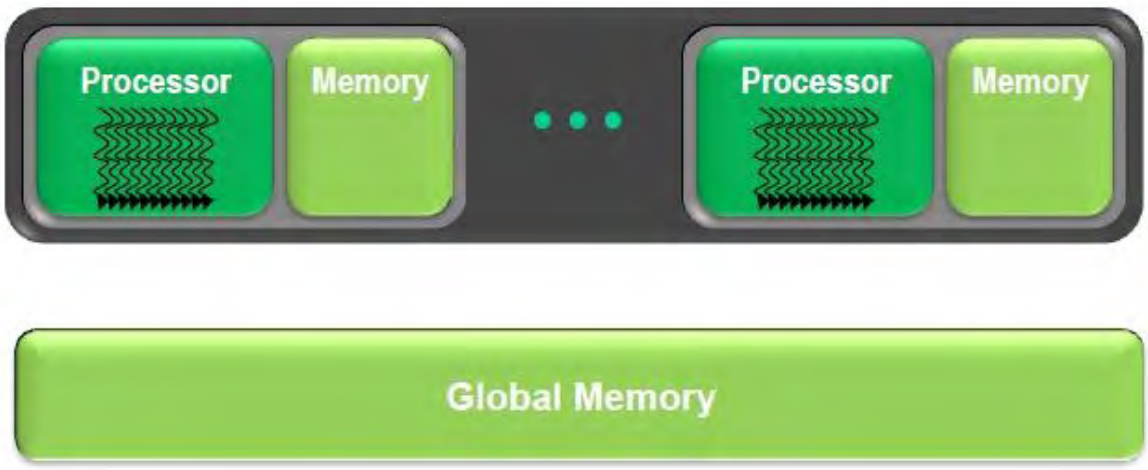
Σχήμα 8 Διαφορές αρχιτεκτονικής CPU και GPU

Η πρώτη σειρά καρτών γραφικών της NVIDIA, η οποία σχεδιάστηκε για το μοντέλο CUDA, ήταν αυτή των G8x. Άλλες νεότερες σειρές καρτών που βγάζει η εν λόγω εταιρία είναι οι GeForce, Tesla, Quadro.

Στο μοντέλο CUDA υπάρχουν οι παρακάτω βασικές έννοιες. Με τον όρο threads αναφερόμαστε στις μονάδες παράλληλης εκτέλεσης / νήματα.Με τον όρο blocks αναφερόμαστε σε ένα σύνολο από threads.Με τον όρο grid κάνουμε λόγο για ένα σύνολο από blocks. Ο αριθμός των διαστάσεων ενός grid μπορεί να είναι μέχρι τρεις και μας δείχνει τον αριθμό των blocks που δεσμεύονται για κάθε διάσταση.

Το συγκεκριμένο μοντέλο χρησιμοποιεί μια επέκταση των γλωσσών C και C++, επιτρέποντας με αυτόν τον τρόπο τον προγραμματισμό συναρτήσεων που μπορούν να εκτελεστούν στη GPU (οι συναρτήσεις αυτές αποκαλούνται kernels).

Οι πολλαπλοί πυρήνες τις GPU συσσωρεύονται σε ένα υψηλότερο επίπεδο διαστρωμάτωσης ως πολυεπεξεργαστές ροής (streaming multiprocessors). Κάθε SM περιέχει 32 πυρήνες CUDA, οι οποίοι μοιράζονται όλοι μαζί από κοινού τους καταχωρητές, τις κρυφές μνήμες, την τοπική μνήμη και τις μονάδες φόρτωσης και αποθήκευσης του SM. Οι 32 πυρήνες cuda που διαθέτει κάθε SM λειτουργούν ως μια ομάδα που μοιράζεται τους ίδιους πόρους και έχει σχεδιαστεί να διαχειρίζεται ταυτόχρονα 32 ίδιες εντολές διαφορετικών δεδομένων από ένα σύνολο 32 νημάτων, το οποίο ονομάζεται warp. Επιπρόσθετα, στις νέες αρχιτεκτονικές, δύο διαφορετικά warps μπορούν να εκτελεστούν ταυτόχρονα στον ίδιο SM, καθώς τώρα ο χρονοπρογραμματιστής (scheduler) του hardware της GPU μπορεί να εισάγει δύο διαφορετικές εντολές ανά χτύπο ρολογιού.

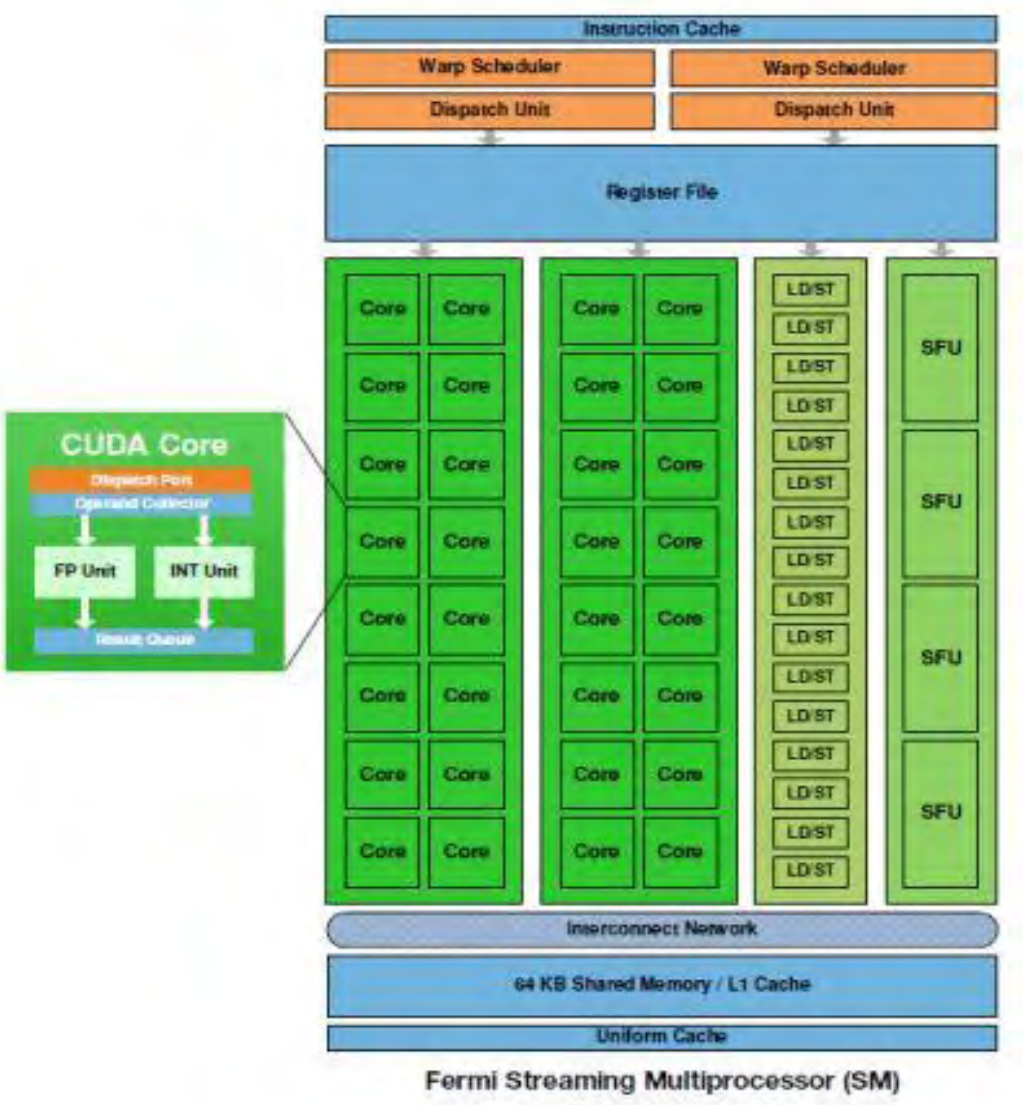


Σχήμα 9 Πολλοί πυρήνες πολλά threads

Τα threads που ανήκουν στο ίδιο block μπορούν να μοιράσουν δεδομένα μεταξύ τους μέσω μια γρήγορης κοινής μνήμης που υπάρχει ανάμεσά τους (shared memory). Επίσης για την ίδια κατηγορία threads, υπάρχουν κατάλληλα φράγματα (barriers) μέσω των οποίων είναι εφικτός ο συγχρονισμός της παράλληλης εκτέλεσής τους. Αντίθετα, threads μεταξύ διαφορετικών blocks, μπορούν να ανταλλάξουν δεδομένα μόνο μέσω μιας αργής κοινής μνήμης (global memory). Για τον συγχρονισμό της δεύτερης κατηγορίας threads δεν υπάρχουν έτοιμες ρουτίνες διαχείρισης που να προσφέρει το μοντέλο και απαιτείται παρέμβαση του προγραμματιστή για την υλοποίηση δικών του ρουτινών συγχρονισμού που θα του αποδώσουν το ζητούμενο αποτέλεσμα.

Η λειτουργία ενός SM μπορεί να συνοψιστεί με τον ακόλουθο τρόπο. Κάθε SM μπορεί να διαχειριστεί μέχρι 48 warps, όπου κάθε warp όπως αναφέρθηκε αποτελείται από μια ομάδα 32 νημάτων. Επομένως, κάθε SM μπορεί να διαχειριστεί μέχρι 1.536 νήματα. Όπως είναι φυσικό με όσα αναφέρθηκαν προηγουμένως, δεν μπορεί να υπάρξει ταυτόχρονη εκτέλεση και των 23.040 νημάτων, αλλά χρειάζεται να υπάρχουν ανά πάσα στιγμή όσο το δυνατόν περισσότερα ενεργά / απασχολημένα νήματα για την επίτευξη μεγάλου παραλληλισμού. Κάτι τέτοιο πρέπει να συμβαίνει διότι όταν ένα warp εκτελεί μια εντολή πρόσβασης σε μνήμη και περιμένει το αποτέλεσμά της, ο χρονοπρογραμματιστής νημάτων (

thread scheduler) την ίδια στιγμή επιλέγει ένα άλλο έτοιμο προς εκτέλεση warp και με αυτόν τον τρόπο κρύβεται / καλύπτεται η καθυστέρηση πρόσβασης. Όλη αυτή η διαδικασία έχει σαν αποτέλεσμα να εκτελείται ανά πάσα στιγμή, ταυτόχρονα, ο μέγιστος αριθμός νημάτων που μπορεί να υποστηρίξει η εκάστοτε GPU.



Σχήμα 10 Streaming Multiprocessor αρχιτεκτονικής Fermi

4.2.2 Παραλληλοποίηση της εφαρμογής μας

Ουσιαστικά η εφαρμογή μας χωρίζεται σε δύο μέρη. Στο δέυτερο μέρος έχουμε την επίλυση ενός συστήματος A*X = B , ενώ στο πρώτο έχουμε την είσοδο των στοιχείων του προβλήματος και την προεργασία των A και B. Η επίλυση του συστήματος γίνεται με τη χρήση μιας συνάρτησης της βιβλιοθήκης Lapack και καταναλώνει έναν σημαντικό μέρος του συνολικού χρόνου. Αυτή η εργασία όμως ασχολείται με το πρώτο μέρος, δηλαδή γίνεται μια προσπάθεια επιτάχυνσης της διαδικασίας παραγωγής των στοιχείων που θα δοθούν ως είσοδοι στη ρουτίνα της Lapack. Κάνοντας profile την εφαρμογή μας (αγνοούμε το κομμάτι της Lapack μιας και αυτός ο χρόνος θα είναι σταθερός) παρατηρούμε ότι η συνάρτηση που υπολογίζει τα εσωτερικά γινόμενα διανυσμάτων(MULT) καλείται επαναληπτικά αρκετές φορές και καταναλώνει σημαντικό κομμάτι του συνολικού χρόνου. Μάλιστα, καλείται μέσω μια άλλης συνάρτησης(ASSENS) η οποία καλείται τόσες φορές όσα, σχεδόν, είναι και τα στοιχεία εισόδου. Η ιδέα είναι αρχικά να μετασχηματίσουμε την MULT ώστε να πραγματοποιεί πολλαπλό υπολογισμό εσωτερικών γινομένων σε κάθε επανάληψη της ASSENS. Αφού τρέξουμε αυτή την υλοποίηση σε περισσότερους του ενός πυρήνες μιας CPU παράλληλα, μπορούμε να αναθέσουμε τους υπολογισμούς αυτούς σε threads μιας GPU και να δούμε αν θα πάρουμε καλύτερους χρόνους ή αν οι πολλαπλές δημιουργίες kernels θα κοστίσουν περισσότερο.

5 Πειραματικά αποτελέσματα

Παρακάτω παρουσιάζονται οι χρόνοι εκτέλεσης της εφαρμογής μας χωρίς τον χρόνο επίλυσης του συστήματος A*X = B, καθώς αυτό υπολογίζεται σε καθορισμένο χρόνο από τη DGBSV( ) :
SINGLE-CORE υλοποίηση,

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self
 time   seconds   seconds    calls      name
 34.78      0.8     0.81    7680000     MULT
 28.26      1.5     0.72   62080300     AINITIA2D
 17.39      1.9     0.40     640000     STIFPE
  8.70      2.1     0.20     640000     ASSENS
  4.35      2.2     0.11    2560000     BUBYMX
  4.35      2.3     0.10     640002     AINITIA
  2.17      2.3     0.01                IINITIA
  0.00      2.3     0.00    7680000     TRANSPOSE
  0.00      2.3     0.00    5120000     BXTXMX
  0.00      2.3     0.00    2560000     ANUTMX
  0.00      2.3     0.00    2560000     BNXTMX
  0.00      2.3     0.00    2560000     BUBXMX
  0.00      2.3     0.00    2560000     BXBYMX
  0.00      2.3     0.00    2560000     BXTYMX
  0.00      2.3     0.00    2560000     BXUMX
  0.00      2.3     0.00    2560000     BYTYMX
  0.00      2.3     0.00    2560000     BYUMX
  0.00      2.3     0.00    2560000     CNYTMX
  0.00      2.3     0.00    2560000     JACOB2
  0.00      2.3     0.00    2560000     SFR2
  0.00      2.3     0.00          2     MODPS
  0.00      2.3     0.00          1     BOUNS
  0.00      2.3     0.00          1     IINITIA2D
  0.00      2.3     0.00          1     intIINITIA
  0.00      2.3     0.00          1     intINITIA2D
```

Σχήμα 11 Σειριακή Υλοποίηση

MULTI-CORE υλοποίηση,

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	name
21.41	0.4	0.41	7680000	gpuMULT
47.26	1.1	0.72	62080300	AINITIA2D
17.39	1.4	0.29	640000	STIFPE
8.32	1.6	0.20	640000	ASSENS
2.70	1.7	0.11	2560000	BUBYMX
2.70	1.8	0.10	640002	AINITIA
0.12	1.8	0.01		IINITIA
0.00	1.8	0.00	7680000	TRANSPOSE
0.00	1.8	0.00	5120000	BXTXMX
0.00	1.8	0.00	2560000	ANUTMX
0.00	1.8	0.00	2560000	BNXTMX
0.00	1.8	0.00	2560000	BUBXMX
0.00	1.8	0.00	2560000	BXBYMX
0.00	1.8	0.00	2560000	BXTYMX
0.00	1.8	0.00	2560000	BXUMX
0.00	1.8	0.00	2560000	BYTYMX
0.00	1.8	0.00	2560000	BYUMX
0.00	1.8	0.00	2560000	CNYTMX
0.00	1.8	0.00	2560000	JACOB2
0.00	1.8	0.00	2560000	SFR2
0.00	1.8	0.00	2	MODPS
0.00	1.8	0.00	1	BOUNS
0.00	1.8	0.00	1	IINITIA2D
0.00	1.8	0.00	1	intIINITIA
0.00	1.8	0.00	1	intINITIA2D

Σχήμα 12 Υλοποίηση σε multicore σύστημα

GPU-CUDA υλοποίηση,

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	name
2.94	0.0	0.04	7680000	gpuMULT
58.45	0.7	0.73	62080300	AINITIA2D
23.09	1.0	0.30	640000	STIFPE
0.53	1.0	0.01	640000	ASSENS
7.22	1.1	0.10	2560000	BUBYMX
7.22	1.2	0.08	640002	AINITIA
0.00	1.2	0.00		IINITIA
0.00	1.2	0.00	7680000	TRANSPOSE
0.00	1.2	0.00	5120000	BXTXMX
0.00	1.2	0.00	2560000	ANUTMX
0.00	1.2	0.00	2560000	BNXTMX
0.00	1.2	0.00	2560000	BUBXMX
0.00	1.2	0.00	2560000	BXBYMX
0.00	1.2	0.00	2560000	BXTYMX
0.00	1.2	0.00	2560000	BXUMX
0.00	1.2	0.00	2560000	BYTYMX
0.00	1.2	0.00	2560000	BYUMX
0.00	1.2	0.00	2560000	CNYTMX
0.00	1.2	0.00	2560000	JACOB2
0.00	1.2	0.00	2560000	SFR2
0.00	1.2	0.00	2	MODPS
0.00	1.2	0.00	1	BOUNS
0.00	1.2	0.00	1	IINITIA2D
0.00	1.2	0.00	1	intIINITIA
0.00	1.2	0.00	1	intINITIA2D

Σχήμα 13 Υλοποίηση σε GPU

Πληροφορίες GPU στην οποία έγιναν οι δοκιμές,

```
Device 0: "GeForce GTX 480"
CUDA Driver Version:                           4.1
CUDA Capability Major/Minor version number:    2.0
Total amount of global memory:                 1536 MBytes (1610285056 bytes)
(15) Multiprocessors x (32) CUDA Cores/MP:     480 CUDA Cores
GPU Clock rate:                                1.40 GHz
Memory Clock rate:                             1848.00 Mhz
Memory Bus Width:                              384-bit
L2 Cache Size:                                 786432 bytes
Max Texture Dimension Sizes                    1D=(65536) 2D=(65536,65535) 3D=(2048,2048,2048)
Max Layered Texture Size (dim) x layers        1D=(16384) x 2048, 2D=(16384,16384) x 2048
Total amount of constant memory:               65536 bytes
Total amount of shared memory per block:       49152 bytes
Total number of registers available per block: 32768
Warp size:                                     32
Maximum number of threads per block:           1024
Maximum sizes of each dimension of a block:    1024 x 1024 x 64
Maximum sizes of each dimension of a grid:     65535 x 65535 x 65535
Texture alignment:                             512 bytes
Maximum memory pitch:                          2147483647 bytes
Concurrent copy and execution:                 Yes with 1 copy engine(s)
Run time limit on kernels:                     No
Integrated GPU sharing Host Memory:            No
Support host page-locked memory mapping:       Yes
Concurrent kernel execution:                   Yes
Alignment requirement for Surfaces:            Yes
Device has ECC support enabled:                No
Device is using TCC driver mode:               No
Device supports Unified Addressing (UVA):      Yes
Device PCI Bus ID / PCI location ID:           1 / 0
```

Σχήμα 14 Χαρακτηριστικά GPU

Όπως παρατηρούμε, στην υλοποίηση σε multicore σύστημα, έχουμε μία ελαφριά επιτάχυνση που οφείλεται στην παραλληλοποίηση της MULT η οποία έχει άμεσο αντίκτυπο στην ASSENS. Όμως, στην υλοποίηση σε GPU, παρατηρούμε πολύ μεγαλύτερη επιτάχυνση στις συναρτήσεις που προαναφέρθηκαν συνεπώς και στο σύνολο του προγράμματός μας. Οι χρόνοι αυτοί είναι αποτέλεσμα εισόδου πίνακα 1,280,000 στοιχείων. Σε πολλές μηχανολογικές εφαρμογές οι είσοδοι μπορεί να είναι πολύ μεγαλύτερες από αυτή που χρησημοποιήσαμε για τις μετρήσεις μας.

6 Επίλογος – Συμπεράσματα

Αυτό που ουσιατικά βλέπουμε είναι ότι έχουμε επιτάχυνση της εφαρμογής μας η οποία εξερτάται σε μεγάλο βαθμό από το πλήθος των στοιχείων εισόδου. Δηλαδή όσο μεγαλύτερη είναι η είσοδος στο πρόγραμμα τόσο περισσότερη θα είναι και η επιτάχυνση που θα βλέπουμε σε σύγκριση με την singlecore αλλά και την multicore υλοποίηση.

Βέβαια, η εφαρμογή μπορεί να επιταχυνθεί ακόμα περισσότερο. Για να έχουμε αυτή τη δυνατότητα πρέπει να εστιάσουμε στην αντικατάσταση της συνάρτησης DGBSV( ) που επιλύει το τελικό σύστημα. Η νέα αυτή συνάρτηση μπορεί να επιδέχεται παραλληλοποίηση σε μεγάλο βαθμό και έτσι να δούμε συνολικά ακόμα μικρότερο χρόνο εκτέλεσης.

7 Βιβλιογραφία

[1] http://www.math.utah.edu/software/lapack/lapack-d/dgbsv.html.

[2] Α. Νικολακάκης, Ανάλυση και σχεδιασμός ηλεκτρικού αισθητήρα γωγμών οδοντωτών τροχών με τη μέθοδο των πεπερασμένων στοιχείων, Αθήνα, 2012.

[3] Λ. Κέππας, Πρόβλεψη μη Γραμμικής Συμπεριφοράς και Διάδοσης Ρωγμής σε Συνθήκες Θερμομηχανικής Κόπωσης με τη Μέθοδο των Συνοριακών Στοιχείων, Πάτρα, 2010.

[4] Ι. Διαμαντάκος, Πρόβλεψη δημιουργίας, διάδοσης και συνένωσης ρωγμών σε αεροπορικά δομικά στοιχεία με πολλαπλή βλάβη, Πάτρα, 2009.

[5] J. Sanders και E. Kandrot, Cuda by Example, NVIDIA.

[6] S. B. Baden, Parallel Computation(lecture slides), 2008.

[7] Χ. Αντωνόπουλος, Συστήματα Υψηλών Επιδόσεων, Βόλος, 2012.

[8] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam και D. Burger, Dark Silicon and the End of Multicore Scaling, ISCA, 2011.

[9] A brief history of microprocessors: http://www.csa.com/discoveryguides/multicore/review2.php.

[10] D. Kirk και W.-m. Hwu, Προγραμματισμός Μαζικά Παράλληλων Επεξεργαστών, 2010.

[11] C. Lin και L. Snyder, Principles of Parallel Programming, Addison Wesley, 2009.

[12] https://developer.nvidia.com/what-cuda.

[13] M. Ashby και K. Johnson, Υλικά και Σχεδιασμός, 2002.

[14] http://en.wikipedia.org/wiki/Fortran.